

## **Documentation for D0JetInfo to be used in the context of Ariel Schwartzman's d0root package.**

### **Introduction**

The d0root package was written by Ariel Schwartzman as a way to analyze D $\emptyset$  data in the root framework. It provides access to three different formats (TMBs, TMBTrees and top\_trees). The D0JetInfo object has been tested exclusively with TMBTrees.

A D0JetInfo object is designed to be used for tagging heavy-flavor jets. Heavy flavor jets will decay and leave either secondary vertices from the b or c-quark hadron decaying or possibly the jet will be tagged with a collinear muon. A D0JetInfo object combines all of the relevant objects to facilitate heavy flavor ID.

A D0JetInfo object starts with a D0Jet. The user may choose which jet algorithm to take, with **coorJCCB** being the default. It also contains those muons within a user-defined ( $\eta, \phi$ ) cone (default 0.5). All of the tracks in the event are used to find a primary vertex, which is attached to the D0JetInfo object. Tracks associated with V0's ( $K^0_L, \Lambda^0$ , etc.) are removed and the remainder are used to find D0TrackJets, which are associated to a D0JetInfo object, if the ( $\eta, \phi$ ) separation between the track jets and calorimeter jets is less than a user defined distance (default 0.7). Along with the track jets come secondary vertices. Muons are also compared to the calorimeter jet direction and associated with the jet if the ( $\eta, \phi$ ) distance is less than 0.5. The event and run number are incorporated into the D0JetInfo object, as are a number of Monte Carlo constructs (if appropriate). If one is running on Monte Carlo data, the leading order parton associated with each of the two leading jets is attached, as are b and c-quark containing hadron decay vertices. Finally, any collinear Monte Carlo muons are associated. Because of problems with the jet energy scale, the jet energy scale is redone (properly including the muon and neutrino correction if there is a collinear muon).

For all of the above variables, one can tune the  $\Delta R$  in ( $\eta, \phi$ ) space, as well as the  $\Delta Z$ . One can set these variables in the D0JetInfoMaker class (discussed below).

The final thing associated with a D0JetInfo object are a series of boolean variables which are set as true or false, depending on whether any of the 12 possible btags\_cert tagging criteria are set. The taggers evaluated are silicon vertex tagger (SVT), soft lepton tag (SLT), Counting Signed Impact Parameter (CSIP) and Jet Lifetime Probability (JLIP). Each of the four taggers has 3 degrees of tightness (Tight, Medium and Loose). The user can turn on or off the evaluation of each of the taggers, although all are turned on by default.

The D0JetInfo object is stored in the d0root\_analysis package. It can be found in d0root\_analysis/objects. The source code will show what is available to the user.

## **DOJETINFOMaker**

The D0JetInfoMaker code exists in a different package. It is in d0root\_jetinfo. A macro which can install the entire package is attached below. The best example use of the code is given in the d0root\_btag package (file d0root\_btag/algorithms/D0JIExample.cpp and .h) The running code for this package is d0root\_btag/runD0JIExample.C. Probably the “Example” method is the best starter, although there are some nice display routines included.

Once one has defined an instance of D0JetInfoMaker (say JI\_Maker), one may modify the internal parameters via the following methods:

**These methods control aspects of D0JetInfo not dealing with btags\_cert (except that you need to send a jet to btags\_cert so as to evaluate whether the taggers fired.**

```
JI_Maker().SetCalJetType("coorJCCB"); which jet algorithm you want
JI_Maker().SetMinMuPt(4.); the minimum Pt of any muon
JI_Maker().SetMinMuQuality(2); the minimum “quality” of the muon
JI_Maker().SetCJPartonDR(1.0);  $\Delta R(\eta, \phi)$  between caljet and parton
JI_Maker().SetCJPartonDZ(100.);  $\Delta Z$  between caljet and parton
JI_Maker().SetCJTJDR(0.7);  $\Delta R(\eta, \phi)$  between caljet and trackjet
JI_Maker().SetCJTJDZ(1.5);  $\Delta Z$  between caljet and trackjet
JI_Maker().SetCJmc2vtxDR(0.5);  $\Delta R(\eta, \phi)$  between caljet and MC 2VTX
JI_Maker().SetCJmc2vtxDZ(3.);  $\Delta Z$  between caljet and MC 2VTX
JI_Maker().SetCJmcmuDR(0.5);  $\Delta R(\eta, \phi)$  between caljet and MC muon
JI_Maker().SetCJmcmuDZ(3.);  $\Delta Z$  between caljet and MC muon
JI_Maker().SetCJrecomuDR(0.5);  $\Delta R(\eta, \phi)$  between caljet and RECO muon
JI_Maker().SetCJrecomuDZ(1.5);  $\Delta Z$  between caljet and RECO muon
```

**These methods govern whether or not one evaluates the named btags\_cert tagger. The default is all are evaluated. You can pay a speed penalty for keeping them all on, so you might consider setting false the ones which you will not use.**

```
JI_Maker().Set_Eval_SVT_Tight(kTRUE);
JI_Maker().Set_Eval_SVT_Medium(kTRUE);
JI_Maker().Set_Eval_SVT_Loose(kTRUE);
JI_Maker().Set_Eval_SLT_Tight(kTRUE);
JI_Maker().Set_Eval_SLT_Medium(kTRUE);
JI_Maker().Set_Eval_SLT_Loose(kTRUE);
JI_Maker().Set_Eval_CSIP_Tight(kTRUE);
JI_Maker().Set_Eval_CSIP_Medium(kTRUE);
JI_Maker().Set_Eval_CSIP_Loose(kTRUE);
JI_Maker().Set_Eval_JLIP_Tight(kTRUE);
JI_Maker().Set_Eval_JLIP_Medium(kTRUE);
JI_Maker().Set_Eval_JLIP_Loose(kTRUE);
```

**The following methods are utilities.**

**JI\_Maker().SetDebug(kTRUE); Turns on extensive internal information reporting  
JI\_Maker().print(); prints the value of all of the internal control parameters.**

**Using the D0JetInfoMaker (\_readEvent is your ReadEvent pointer and \_data\_type is either “MC” or “DATA”). If will fill a TObjArray containing D0JetInfo objects.**

**JI\_Maker().Make(\_readEvent,\_data\_type); Fill TObjArray of D0JetInfo objects  
TObjArray myJetInfo = JI\_Maker().GetD0JetInfo(); Access TObjArray  
myJetInfo.SetOwner(kTRUE); Necessary to suppress memory leaks.**

## **D0JetInfo data access methods.**

Assume that you have a D0JetInfo object called jInfo. You can then get access to the associated objects via the following methods.

### **Raw Associated Objects Access**

D0MCParticle*	parton	= jInfo->GetParton();
D0Jet*	cj	= jInfo->GetCalJet();
D0TrackJet*	tj	= jInfo->GetTrackJet();
D0Vertex*	pv	= jInfo->PrimaryVertex();
int	run	= jInfo->Get_runnum();
int	evt	= jInfo->Get_evtnum();
TObjArray	muons	= jInfo->GetMuons();
TObjArray	MCmuons	= jInfo->GetMCMuons();
TObjArray	MCvtx2	= jInfo->Get_MC_2Vtx();
TObjArray	vtx2	= jInfo->Get_Vtx();

### **btags\_cert booleans for each of the various taggers**

bool	svt_tight	= jInfo->GetBT_SVT_Tight();
bool	svt_medium	= jInfo->GetBT_SVT_Medium();
bool	svt_loose	= jInfo->GetBT_SVT_Loose();
bool	slt_tight	= jInfo->GetBT_SLT_Tight();
bool	slt_medium	= jInfo->GetBT_SLT_Medium();
bool	slt_loose	= jInfo->GetBT_SLT_Loose();
bool	csip_tight	= jInfo->GetBT_CSIP_Tight();
bool	csip_medium	= jInfo->GetBT_CSIP_Medium();
bool	csip_loose	= jInfo->GetBT_CSIP_Loose();
bool	jlip_tight	= jInfo->GetBT_JLIP_Tight();
bool	jlip_medium	= jInfo->GetBT_JLIP_Medium();
bool	jlip_loose	= jInfo->GetBT_JLIP_Loose();

**Access to jet kinematics with proper jet energy scale (including muons, if appropriate). If there is no muon, this should be the same as the standard calorimeter jet Pt, modulo JES versioning.**

TLorentzVector*	ji_p	= jInfo->GetP();
double	ji_pt	= jInfo->GetPt();

### **Access to some of the more internal variables**

double	cjpt	= jInfo->CalJet_Pt();
double	cjeta	= jInfo->CalJet_Eta();
double	cjphi	= jInfo->CalJet_Phi();
TLorentzVector	cjLV	= jInfo->CalJet_4vec();
double	tjpt	= jInfo->TrackJet_Pt();

```

double          tjeta      = jInfo->TrackJet_Eta();
double          tjphi     = jInfo->TrackJet_Phi();

bool           central    = jInfo->IsCentral();           |cjeta| < 0.5
bool           forward   = jInfo->IsForward();  2.0 < |cjeta| < 2.5
bool           ismutagged = jInfo->IsMuonTagged(int QUAL);

has collinear muon with quality > QUAL
bool           is2vtxtagged = jInfo->Is2VTXTagged(int TYPE);
TYPE = 1 means a secondary vertex on the same side of the primary vertex
as the jet is going. In addition, the 2D decay length must be less
than 2.5.
TYPE = 2 as TYPE 1, plus the 2D decay length significance must exceed 32.
bool           ismcmutagged = jInfo->IsMuonTagged();

has collinear Monte Carlo muon
bool           ismu_in_list = jInfo->Muon_From_VTX_List(D0Muon* mu)
specified muon is attached to a RECO 2vtx associated with this jet
bool           vtx_w_muon  = jInfo->VTX_contains_muon(D0Vertex* vtx)
specified vertex has a muon as one of the tracks.

double          DecLen2D   = jInfo->Bst2VTX_2D_DecayLength()
the 2D decay length for the best secondary vertex (best secondary vertex has
largest 3D decay length significance, a 2D decay length less than 2.5 cm and
the vertex on the same side of the primary vertex as the direction the jet is
going)
double          DecLenSig2D = jInfo->Bst2VTX_2D_DecayLengthSignificance()
the 2D decay length significance for the best secondary vertex
double          DecLen3D   = jInfo->Bst2VTX_3D_DecayLength()
the 3D decay length for the best secondary vertex
double          DecLenSig3D = jInfo->Bst2VTX_3D_DecayLengthSignificance()
the 3D decay length significance for the best secondary vertex

int            caljetqual = jInfo->CalJetQuality()
the quality rating for the calorimeter jet.
int            LOPartonID = jInfo->LOPartonID()
if the event is Monte Carlo, this is the PDG parton ID (best guess).
bool           isbVTX    = jInfo->bVerte(D0SimpleVertex* vtx)
if the event is Monte Carlo, this tags the vertex ‘vtx’ as coming from a
b-hadron
bool           iscVTX    = jInfo->cVerte(D0SimpleVertex* vtx)
if the event is Monte Carlo, this tags the vertex ‘vtx’ as coming from a
c-hadron

double          CosThetaStar = jInfo->CosThetaStar(D0JetInfo* ji2)
calculates the cosine(theta star) (i.e. the angle between jInfo and the beam
direction in the CM frame of jInfo and ji2)
double          DiJetMass   = jInfo->Mass(D0JetInfo* ji2)

```

**calculates the invariant mass of the two objects (jInfo & ji2)**

## **APPENDIX (Install Macro for d0root\_ with d0root\_jetinfo objects)**

```
# This is verified with D0RunII p14.05.02. To modify to run with p16,
# read carefully and make changes where noted.
#
#Uncomment the following for a p16 build
#-----
#setup D0RunII p16.00.00
#addpkg tmb_tree

d0setwa
setup d0cvs

#Install packages
#-----
addpkg d0root_analysis v00-09-51
addpkg d0root_tmbtree v00-09-25
addpkg d0root_btag v00-09-60

# If p16, comment out the next two lines.
#-----
addpkg tmb_tree p14-br-20
addpkg tmb_analyze p14-br-16

addpkg jetcorr v05-01-00

#Only needed for BphysicsAnalysis
#-----
#addpkg -h AATrack

#Generate TMBTreeClasses file
#-----
cp tmb_analyze/macros/MakeTMBTreeClasses_so.C .
root -l -q MakeTMBTreeClasses_so.C

# In order to use the D0JetInfo objects (and maker), remove a single "#"
# in front of each of the following 4 lines.
#Compile jetcorr
#-----
cd jetcorr
root -l -q MkRootLib.C
cd ..
addpkg btags_cert v00-01-02
addpkg bc_csiptagger v00-00-09
```

```
addpkg d0root_csip v00-00-12
addpkg d0root_jlip v00-00-07
addpkg d0root_slt v00-00-04
addpkg d0root_jetinfo v00-01-04

#Compile d0root packages
#-----
gmake d0root_btag.all

# Pick if you are using p14 or p16 TMBTree code
# Note if you don't pick one, it won't compile
#-----
cd d0root_tmbtree/d0root_tmbtree/
cp version_p14.hpp version_dependent_include.hpp
#cp version_p16.hpp version_dependent_include.hpp
cd ../.

#-----
# Test
#-----
#root d0root_btag/testTMBtree.C

# To test jetinfo objects, substitute following line for previous line
root -l d0root_btag/runD0JIEExample.C
```